## Research note 33

# The Instruction Subset Hypothesis

Edward McDaid & Sarah McDaid

23 Jun 2024

**Instruction subsets and related approaches are underpinned by a hypothesis that making generated code more closely resemble that produced by humans also reduces the effort required to find solutions. The results of several studies provide dramatic confirmation of this idea, however it's still not totally clear why this is so**.

The instruction subset approach is the gift that keeps on giving. The original incarnation produced reductions in the size of the inductive programming search space of tens of orders of magnitude. While these results were both surprising and welcome, the scale of the reductions hinted that further significant gains might also be achievable.

Instruction subsets are an answer the question of what combinations of instructions can be used together to form solutions? In this case the answer takes the form of many overlapping subsets of instructions - extracted from a large body of human code. Each subset represents a different prediction of the combination of instructions required to produce a solution to any problem. This discovery quickly led to another very simple idea.

Instruction digrams describe the ways in which these instructions can be combined to form solutions. This is achieved simply by counting the number of occurrences of one instruction calling another - again, as observed in a large code sample. For a given instruction subset, this information can be used to further constrain the possibilities that need to be considered as candidate solutions are assembled. Instruction digrams can further reduce the search space by another 5 orders of magnitude.

Papers on both of these approaches have been published but each of these only describes a snapshot of the work as it existed several months prior to the date of publication. Things have moved on considerably at Zoea since then. Instruction subsets have evolved through several iterations with a new clustering algorithm and optimisation of the deployment approach. Instruction digrams has seen the development of a better parser that is capable of handling a wider range of programming languages. There have also been further successful studies involving related but different approaches. Some of these results will also be published in due course.

But why do any of these approaches actually work? If we think in terms of black box behaviour, there are often many ways to write any given program. Some of these variations will have different characteristics with regard to their size or efficiency, for example. However, they are all equally valid with respect to a given input-output specification.

At the same time much human code has various degrees of self-similarity. Individual developers and teams have their own styles - at least some of which reflects whatever best practises are currently in fashion. In addition, globalisation and the internet mean that views on what constitutes good code have become increasingly homogenous. This all means that software development is very much a human culture, verging on a religion. People could write really weird code and indeed some do so for fun, but they would struggle to hold down a job if they did so at work.

Odd-looking, non-human style code solutions will exist for every problem, and there will be very many of them. Most of these will be very different from the equivalent human code solutions. They will also mostly be very different from one another. In the search space of candidate solutions, the human examples are, figuratively speaking, all huddled together in one corner, while the code that doesn't look human-like can be found everywhere else.

It should not be surprising therefore, that if we attempt to understand the characteristics of human code in terms of what instructions are used together or what instructions call other instructions, then human produced examples are both distinct from random noise and also vanishingly less common. Furthermore, human developers generally aim to produce the simplest solution to any problem. This suggests that looking for code that is human-like is more likely to turn up a solution. Since the space of human-like code is very much smaller than that for non-human-like code, then we should come across a solution more quickly.

So, do developers brains, deep down, contain constructs similar to instruction subsets and digrams? There is probably some sort of psychology experiment that could answer this question, but in the end it doesn't really matter either way. In the same way that it doesn't matter whether human brains actually work like neural networks, the end result is still worth having.

Learn more at **zoea.co.uk**